Уральский федеральный университет имени первого Президента России Б.Н.Ельцина	Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина» (УрФУ) ИРИТ-РТФ Базовая кафедра «Аналитика больших данных и методы видео анализа»

Grade on project
____
Project supervisor
______SAIF M.A.___
Commission's members
____
____
____

Defense date
________21.06.2025_____

Personal Finance management Web Application

Student's name: SIDI MOHAMED LEMIN
Group: RIM-140930

Yekaterinburg
2025

Content

Table des matières

Executive Summary

The FinTrack+ project is a comprehensive personal finance management web application developed using Django, a high-level Python web framework. The primary objective of this project is to provide users with an intuitive platform to track their income, monitor expenses, manage monthly budgets, and visualize financial trends through interactive charts and summaries. It empowers individuals to gain insights into their financial behavior and make informed decisions.

Technologically, FinTrack+ integrates several modern tools and libraries including Django Allauth for secure user authentication, Crispy Forms and Bootstrap for elegant and responsive form handling, Chart.js for real-time data visualization, and SQLite (with Render-ready PostgreSQL configuration) for reliable data storage. The system architecture follows Django's Model-View-Template (MVT) paradigm, ensuring maintainability and separation of concerns.

Key accomplishments of the project include the successful implementation of core financial features (income/expense tracking, budget setting, dynamic dashboards), secure registration and login systems with email confirmation, recurring transaction processing via custom Django management commands, and export capabilities for generating PDF and Excel reports. The platform is also designed with multi-device compatibility in mind, ensuring seamless access across desktops and mobile browsers.

Overall, FinTrack+ represents a practical and scalable solution for personal financial management, with room for future enhancements such as real-time notifications, advanced analytics, and AI-based spending suggestions.

Introduction

With the rapid development of digital technologies and software solutions, the need for effective tools to manage personal finances is more urgent than ever. With multiple sources of income and a variety of daily expenses, individuals find themselves in dire need of a system that helps them track their financial transactions, analyze their spending pattern, and set clear and thoughtful financial goals. This project aims to design and develop a comprehensive web application for personal budget management using the popular Django framework.

This application provides a secure interactive environment for users, where they can record income and expenses, set monthly budgets, identify recurring financial transactions such as rent or salaries, and display dynamic visual reports that highlight the financial distribution by categories. The system has an integrated authentication and account creation system, with support for email confirmation and password reset, ensuring a reliable and secure user experience.

This project is characterized by its careful structural organization in terms of dividing it into independent sub-applications within the main project, which facilitates future development and expansion. Best practices in interface design using Bootstrap and the integration of the Chart.js library to display data in an attractive and clear way, making this app an ideal tool for managing personal finances for both individual users and small families.

Project Objective

This project aims to develop a secure and easy-to-use web application that enables individuals to take full control of their personal finances. The system allows users to seamlessly manage their income, expenses, and monthly budgets through a simple and intuitive interface. By providing interactive graphs and automated features such as recurring transactions and over-budget alerts, the app helps users enhance financial awareness and adopt more disciplined spending habits. The system is designed to simplify the process of tracking money while maintaining data privacy and ease of use, both for beginners and experienced users.

Project Architecture

The development of FinTrack_Project has adopted a flexible and carefully organized architecture by applying the principle of Separation of Concerns, which is one of the main pillars of designing scalable and maintainable systems. The project is based on the Django framework, taking advantage of its ability to divide components into independent applications (Apps) that facilitate collective development and future updates.

Explanation of the Django structure:

The Project is based on a strict hierarchical and structural organization that respects the principle of separation between logical, visual, and graphical components, which facilitates development, maintenance, and testing. The project is divided into main and sub-folders, with each component having its own specific function within the overall system architecture.

In Django, dividing the project into Apps is a standard approach to code organization, with each App dedicated to handling a specific part of the system's logic. For example, one app is dedicated to managing finances, another to authentication and login, and a third to managing user profiles.

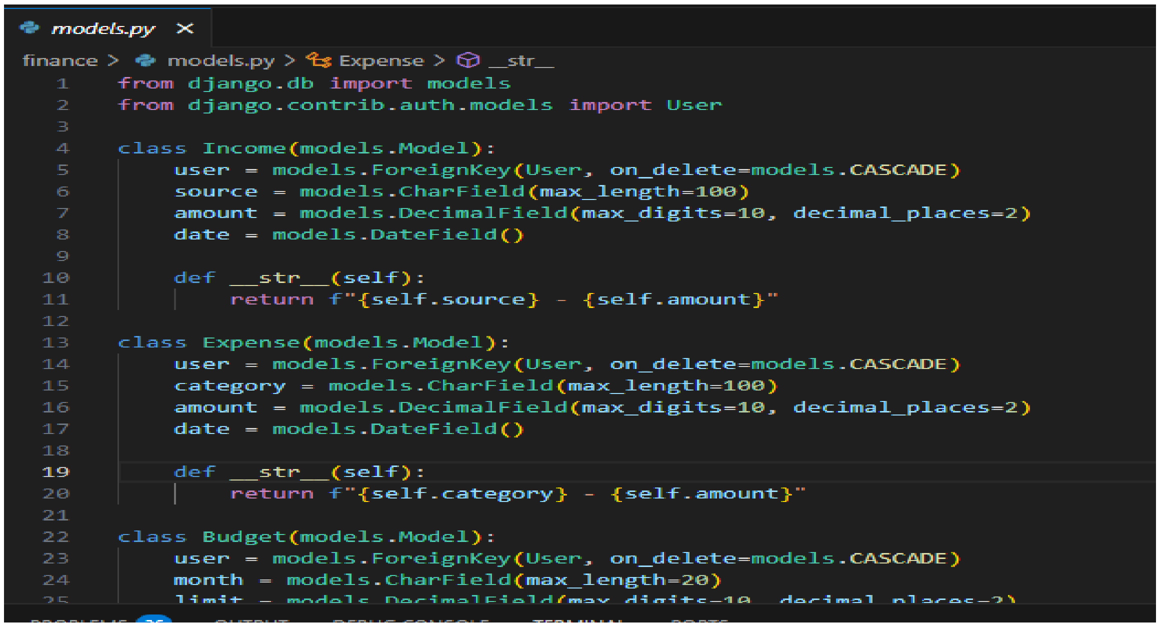
Each application contains core files and functions such as

a) models.py: defines the data models associated with the database.

This file contains the basic forms associated with the database, which represent the main system entities, including:

- Transaction: Used to store financial transaction information, whether income or expense, and includes fields such as Date, Type, Amount, and Notes.

- RecurringTransaction: A form dedicated to recording periodically recurring financial transactions (such as subscriptions or monthly salaries), with the ability to specify the type of recurrence and the time period



```python
from django.db import models
from django.contrib.auth.models import User

class Income(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    source = models.CharField(max_length=100)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    date = models.DateField()

    def __str__(self):
        return f"{self.source} - {self.amount}"

class Expense(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    category = models.CharField(max_length=100)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    date = models.DateField()

    def __str__(self):
        return f"{self.category} - {self.amount}"

class Budget(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    month = models.CharField(max_length=20)
    limit = models.DecimalField(max_digits=10, decimal_places=2)
```

b) views.py: Contains the control logic (controlling how requests are processed).

The views.py file within the finance/ application serves as a crucial component in the Controller layer of the FinTrack+ architecture, following the Model-View-Controller (MVC) design pattern. It is responsible for handling the logic that connects the user interface (templates) with the data layer (models) through well-structured functions (views) that respond to user actions.

The main responsibilities of this file include:

- Displaying the Financial Dashboard:

Presents users with a comprehensive overview that includes total income, total expenses, remaining budget, and dynamic visual charts powered by Chart.js.

- Adding Financial Transactions:

Handles form submissions for new income or expense entries. It validates user input, processes the POST request, and commits the data to the database securely.

- Editing and Deleting Transactions:

Provides functionality to modify or remove existing records, with proper permission checks to ensure user authorization.

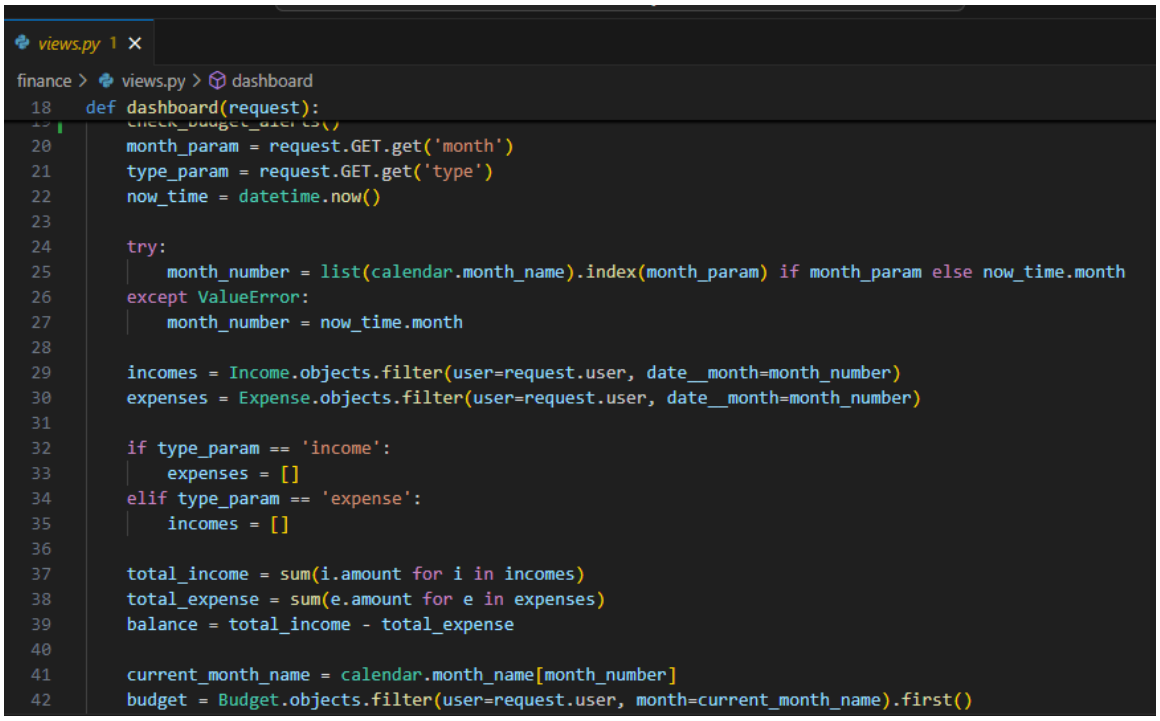
- Filtering Data by Month or Type:

Implements filtering logic to allow users to view transactions based on a selected month or type (income or expense), enhancing precise financial tracking.

• Automated Handling of Recurring Transactions:

Works in conjunction with custom management commands to periodically process recurring entries without manual intervention.

Each of these functions is tightly integrated with user-friendly interfaces, real-time data feedback, and messaging systems, ensuring a responsive and reliable user experience within the application.

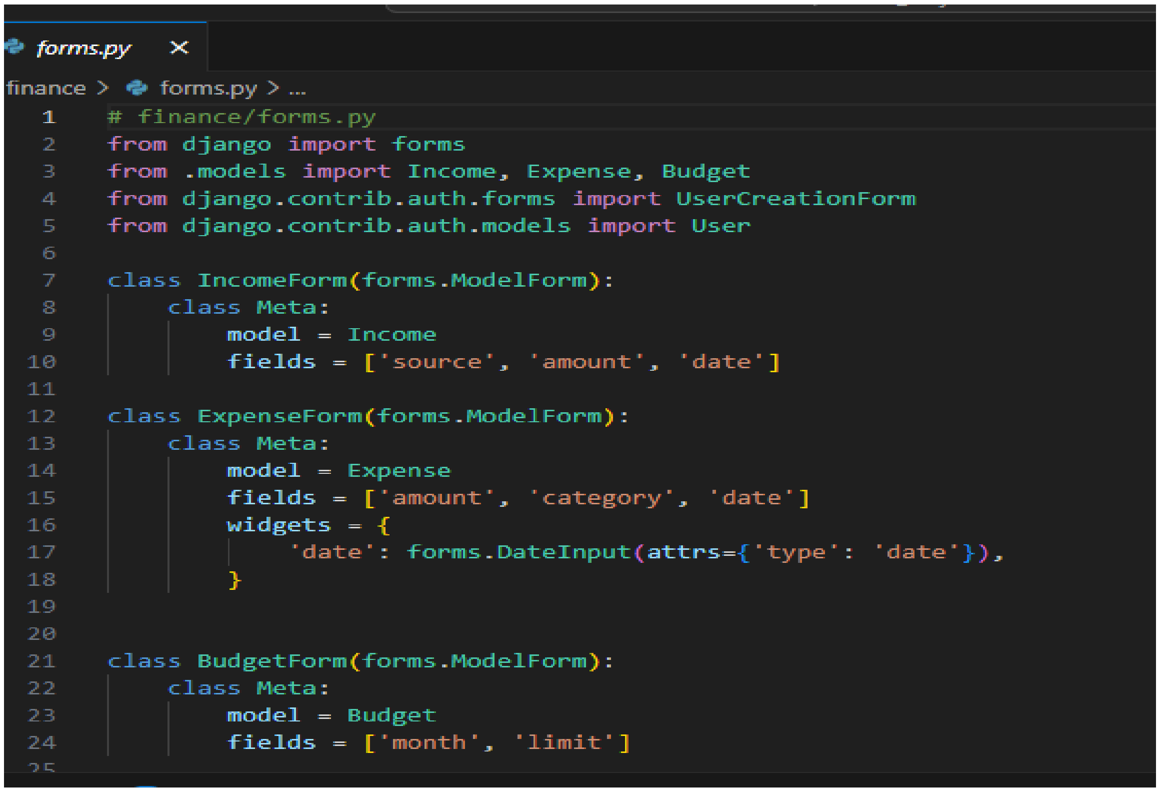


```python
def dashboard(request):
    check_budget_alerts()
    month_param = request.GET.get('month')
    type_param = request.GET.get('type')
    now_time = datetime.now()

    try:
        month_number = list(calendar.month_name).index(month_param) if month_param else now_time.month
    except ValueError:
        month_number = now_time.month

    incomes = Income.objects.filter(user=request.user, date__month=month_number)
    expenses = Expense.objects.filter(user=request.user, date__month=month_number)

    if type_param == 'income':
        expenses = []
    elif type_param == 'expense':
        incomes = []

    total_income = sum(i.amount for i in incomes)
    total_expense = sum(e.amount for e in expenses)
    balance = total_income - total_expense

    current_month_name = calendar.month_name[month_number]
    budget = Budget.objects.filter(user=request.user, month=current_month_name).first()
```

c) **forms.py:** for creating and validating input forms.

The forms.py module plays a crucial role in managing and validating user input across the application. It defines custom Django forms for both the finance and user-related operations, including income and expense submission, profile updates, and user registration. These forms not only provide structure for collecting data but also ensure data integrity through validation constraints, custom clean methods, and styling integration using Django Crispy Forms.

By decoupling form logic from views, the application achieves better modularity and maintainability. Additionally, using forms enhances user experience with features like real-time error display and automatic rendering of fields, ensuring the application's reliability when interacting with user-provided data.

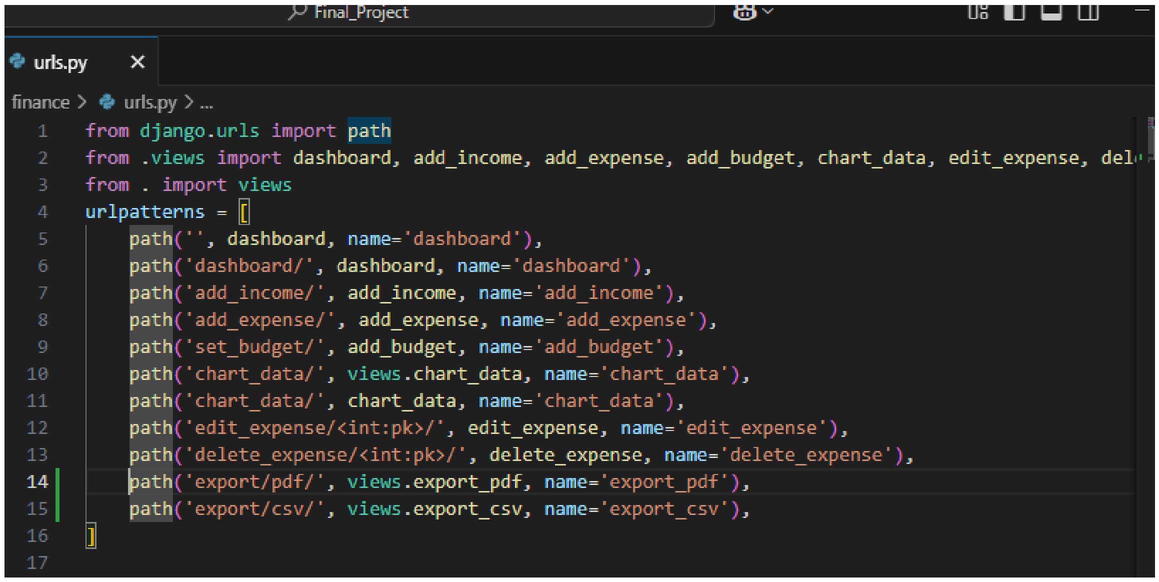


```python
# finance/forms.py
from django import forms
from .models import Income, Expense, Budget
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class IncomeForm(forms.ModelForm):
    class Meta:
        model = Income
        fields = ['source', 'amount', 'date']

class ExpenseForm(forms.ModelForm):
    class Meta:
        model = Expense
        fields = ['amount', 'category', 'date']
        widgets = {
            'date': forms.DateInput(attrs={'type': 'date'}),
        }


class BudgetForm(forms.ModelForm):
    class Meta:
        model = Budget
        fields = ['month', 'limit']
```

d) urls.py: Defines the application's navigation paths.

The urls.py files in each Django app define the URL patterns that determine how the application responds to incoming HTTP requests. In this project, both the finance/urls.py and users/urls.py files serve as routing configurations for their respective domains. They map specific URL paths to their corresponding views, enabling clear and intuitive navigation such as /finance/dashboard/, /income/add/, or /account/login/.

The main project-level urls.py file (within fintrack_project/) includes and organizes these routes using Django's include() function. This modular URL design aligns with Django best practices, allowing each app to remain self-contained while contributing to the global route structure of the FinTrack+ application.



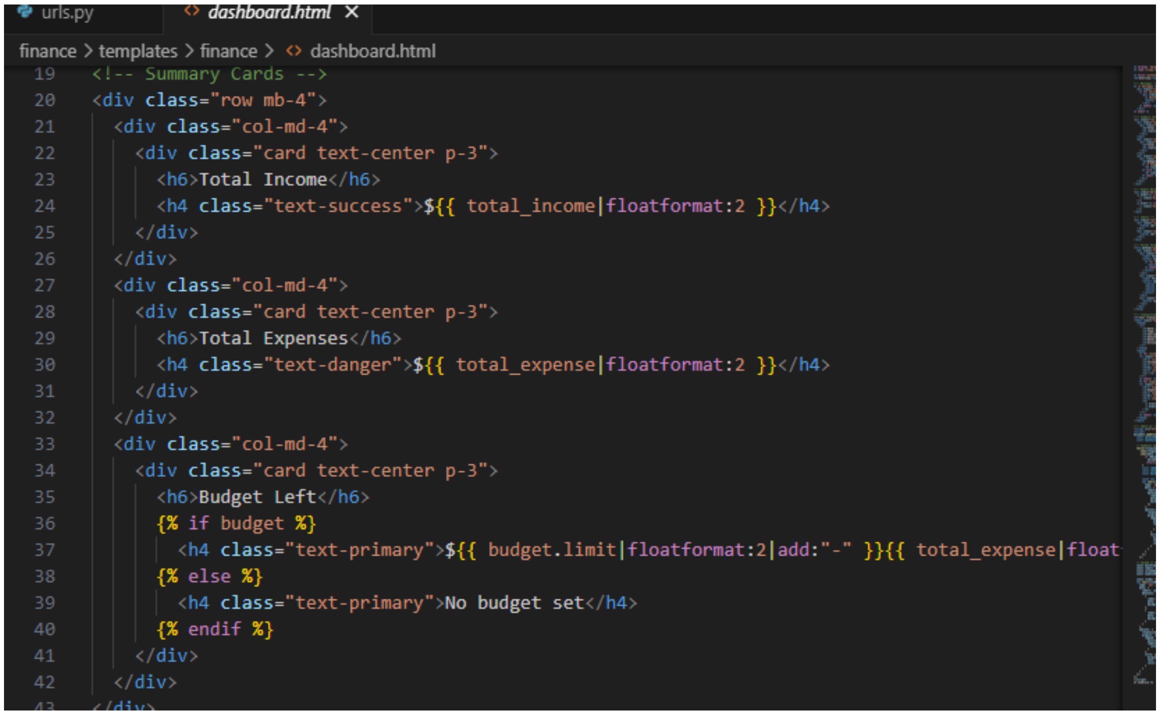
```python
from django.urls import path
from .views import dashboard, add_income, add_expense, add_budget, chart_data, edit_expense, del
from . import views
urlpatterns = [
    path('', dashboard, name='dashboard'),
    path('dashboard/', dashboard, name='dashboard'),
    path('add_income/', add_income, name='add_income'),
    path('add_expense/', add_expense, name='add_expense'),
    path('set_budget/', add_budget, name='add_budget'),
    path('chart_data/', views.chart_data, name='chart_data'),
    path('chart_data/', chart_data, name='chart_data'),
    path('edit_expense/<int:pk>/', edit_expense, name='edit_expense'),
    path('delete_expense/<int:pk>/', delete_expense, name='delete_expense'),
    path('export/pdf/', views.export_pdf, name='export_pdf'),
    path('export/csv/', views.export_csv, name='export_csv'),
]
```

e) Templates: Dynamic HTML Rendering and UI Presentation

The templates/ directory is a central component of the Django project's front-end architecture. It contains all the HTML files used to render views dynamically based on context data passed from the backend. Within the finance/ and users/ apps, this folder includes template files such as dashboard.html, income_form.html, register.html, and login.html, which collectively form the user interface of the FinTrack+ system.

These templates are written using Django's templating language, allowing the embedding of dynamic content such as user-specific transactions, income summaries, budget charts, and form validation messages. They leverage reusable components (like base.html) to maintain consistent layout and design across pages. Furthermore, integration with Bootstrap and Crispy Forms ensures that the templates are not only functional but also responsive and visually appealing.

The separation of presentation logic into the templates/ folder promotes clean code architecture by distinguishing backend logic from frontend layout, thereby aligning with the Model-View-Template (MVT) design pattern fundamental to Django.



```html
<!-- Summary Cards -->
<div class="row mb-4">
  <div class="col-md-4">
    <div class="card text-center p-3">
      <h6>Total Income</h6>
      <h4 class="text-success">${{ total_income|floatformat:2 }}</h4>
    </div>
  </div>
  <div class="col-md-4">
    <div class="card text-center p-3">
      <h6>Total Expenses</h6>
      <h4 class="text-danger">${{ total_expense|floatformat:2 }}</h4>
    </div>
  </div>
  <div class="col-md-4">
    <div class="card text-center p-3">
      <h6>Budget Left</h6>
      {% if budget %}
        <h4 class="text-primary">${{ budget.limit|floatformat:2|add:"-" }}{{ total_expense|float
      {% else %}
        <h4 class="text-primary">No budget set</h4>
      {% endif %}
    </div>
  </div>
</div>
```

```
Final_Project/
│
├── finance/              # Handles financial logic: income, expenses, budgets
│   ├── models.py          # Defines models like Income, Expense, Budget
│   ├── views.py          # Contains view functions for financial operations
│   ├── forms.py          # Forms for creating and validating financial entries
│   ├── urls.py          # URL patterns specific to finance operations
│   ├── templates/        # HTML templates related to finance app
│   └── management/
│       └── commands/      # Custom Django commands (e.g., process_recurring.py)
│
├── account/              # User authentication: login, registration, password reset
│   ├── views.py          # Authentication and email verification logic
│   ├── urls.py          # Routes for login, signup, etc.
│   └── templates/        # Auth-related frontend templates
│
├── users/               # Manages user profile and account settings
│   ├── models.py          # CustomUser or UserProfile model
│   ├── views.py          # Views for displaying/editing user data
│   └── templates/       # Templates for profile pages
│
├── templates/           # Shared base layout and general templates
├── static/              # CSS, JavaScript, Chart.js, and static assets
├── media/               # User-uploaded files (e.g., profile pictures)
├── db.sqlite3           # SQLite database for development
├── manage.py            # Django's command-line utility
└── requirements.txt     # List of required Python packages
```

Design Methodology

The project was developed using a systematic design methodology that aims for efficiency, scalability, and long-term maintainability. This methodology is based on a set of proven engineering principles, which are the cornerstones of building professional web systems using the Django framework.

I. Separation of Concerns (SoC)

This principle was applied by distributing the project logic into independent applications (Apps), where each module is responsible for a specific aspect of the system:

Finance app deals only with financial processes.

Account app is dedicated to authentication and access management.

Asers app for managing user accounts and profiles.

This separation makes the project more organized and makes it easier to expand in the future without affecting the other components.

II. Using the modular architecture for Django projects

The recommended architecture in Django was adhered to, which separates between:

Models: To define the data structure.

Views: To process requests and return responses.

Templates: To display data to users.

Routes (URLs): To route requests to the appropriate interfaces.

III. Adopting Reusable Components

In an effort to develop flexible, standardized, and easy-to-maintain interfaces, Reusable Components has been adopted as a key methodology in the design of user interfaces. This was done by creating a standardized master template known as base.html, which serves as the skeleton for all front-end pages within the system.

This base template includes elements that are common across all pages of the project, such as:

Header

Navigation Bar

General Container Content

Footer

Include basic CSS/JS files

The rest of the application sub-templates (e.g. login page, dashboard, add transaction page, etc.) are built on the basis of inheriting this main template using Django's template inheritance mechanism via {% extends ‘base.html’ %}. Modifiable areas within the base template are defined using blocks such as {% block content %}, which allows each page to customize its internal content without having to rewrite the entire structure.

This approach offers several advantages, most notably:

DRY - Don't Repeat Yourself, which reduces code complexity.

Easy maintenance and modernization: Any change made to the overall design can be reflected on all pages immediately by modifying the basic template only.

Improved user experience (UX) by maintaining a consistent and standardized format throughout the system.

The same template can be reused in multiple sub-applications, saving time and effort during the development process.

Academically, this approach is a real-world example of applying Clean Design Principles in a web environment, which enhances the quality and scalability of the software product.

IV. Dynamic & Interactive Features

One of the most important components of the success of modern web applications is their ability to provide an interactive experience for the user, beyond the traditional concept of static pages towards flexible interfaces that respond to user input and display data visually and attractively. In this context, the project was keen to include dynamic features based on the use of the Chart.js library to display interactive graphs, especially within the Dashboard page.

These features work by combining HTML interfaces with JavaScript files, where an asynchronous AJAX Request is sent from the browser to the server to obtain expense data based on the selected month. This data is then converted to JSON format, which is then plotted directly in the browser using the Chart.js library as Pie Charts or other graphical styles.

This interactivity in data visualization is a real-world application of Event-Driven Programming, which is a recent trend in user interface development, especially when it comes to designing financial dashboards.

The pedagogical and technical benefits of adopting this methodology include

- ➢ Enhanced user experience (Enhanced UX) by providing an interactive visual interface.
- ➢ Enabling the user to visually analyze their financial data, which facilitates decision-making.
- ➢ Separate the data processing logic from the display interface, which enhances reusability and extensibility.
- ➢ Take advantage of modern browser capabilities without the need to reload the page (Asynchronous Behavior).

Academically, the inclusion of these features is an actual example of the integration of backend and frontend programming into a single application, highlighting the importance of integrated skills in the development of modern web systems.

Security and Verification

Security is one of the essential pillars in developing web applications, especially those that deal with financial data and sensitive user information. Therefore, FinTrack is designed according to a methodology that pays great attention to implementing multi-level security and verification procedures to ensure data confidentiality, prevent unauthorized access, and enhance user confidence in the system.

1. Authentication System

The project relies on the Django framework's built-in authentication system, which provides strong user authentication mechanisms. The login, logout, and registration interfaces are customized, and the user experience is improved by designing custom pages in a clear and easy-to-use format.

2. Email Verification

An email verification feature has been included after registration, where the user can only activate their account after clicking on the activation link sent to their email. This is an effective tool to protect the system from fake accounts or random registration attempts, and enhances the credibility of registered accounts.

3. Password Reset

The project provides a secure and flexible password reset mechanism where the user can request a reset link to be sent to their email, allowing them to choose a new password in a way that keeps their data confidential. This feature includes customized forms and templates for the user interface.

4. Session Security

The project uses temporary Sessions managed by the server to ensure the security of the user's interaction with the system. Sessions are closed when logged out or after a timeout, preventing snooping on accounts that are open for long periods of time without use.

5. Protection against common attacks

By default, Django provides built-in protection against several common threats such as:

- CSRF (Cross-Site Request Forgery): By including protection codes in every form.
- XSS (Cross-Site Scripting): By automatically filtering inputs.
- SQL Injection: By using ORM instead of writing manual SQL queries.

The adoption of these procedures is a direct application of the concepts of Software Security Engineering, which makes the project suitable for use in real contexts, as well as an educational platform for applying security theories in a real-world environment.

Comprehensive overview of the dashboard and transaction Log

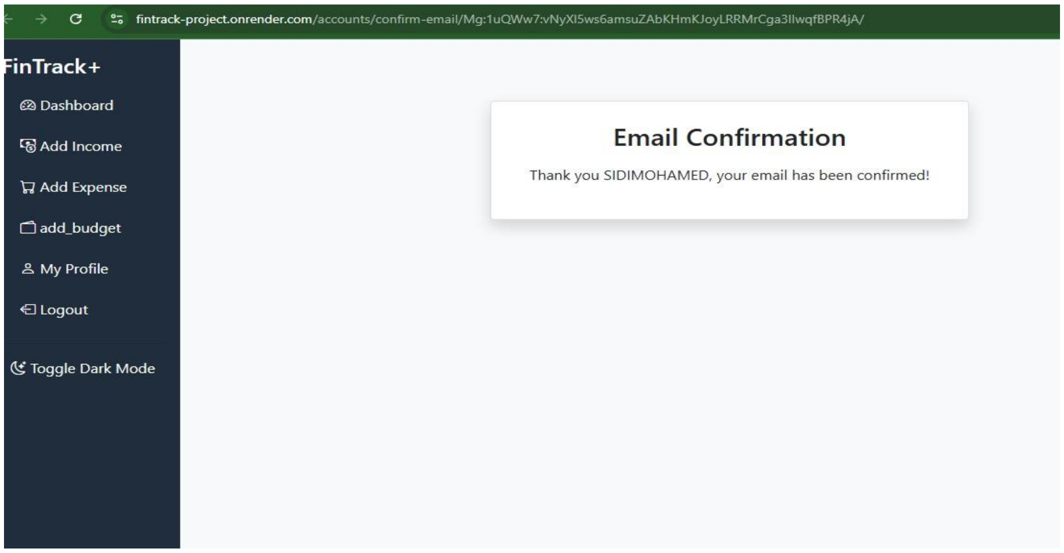
The dashboard in FinTrack+ provides users with a powerful and user-friendly interface that summarizes their financial status in real-time. It showcases an aggregated view of total income, total expenses, and the current budget balance using clear and dynamic overview cards. These visual indicators help users immediately assess whether they are within their set budget or have exceeded it. In addition to numeric summaries, the system integrates a visually rich pie chart using the Chart.js library, which categorizes expenses based on their source (e.g., rent, food, travel), allowing for quick and intuitive financial analysis.

Alongside the dashboard, the transaction log plays a vital role by listing all income and expense records chronologically. This unified ledger includes essential fields such as date, type, category/source, amount, and action buttons for editing or deleting individual entries. By combining visualization with transactional granularity, this section fosters transparency, enhances control over spending behavior, and supports better-informed financial decisions.

From a technical perspective, these features exemplify the effective integration of Django's backend capabilities with modern frontend libraries, resulting in an interactive and scalable solution for personal financial management.

Screenshots (Visual Evidence):

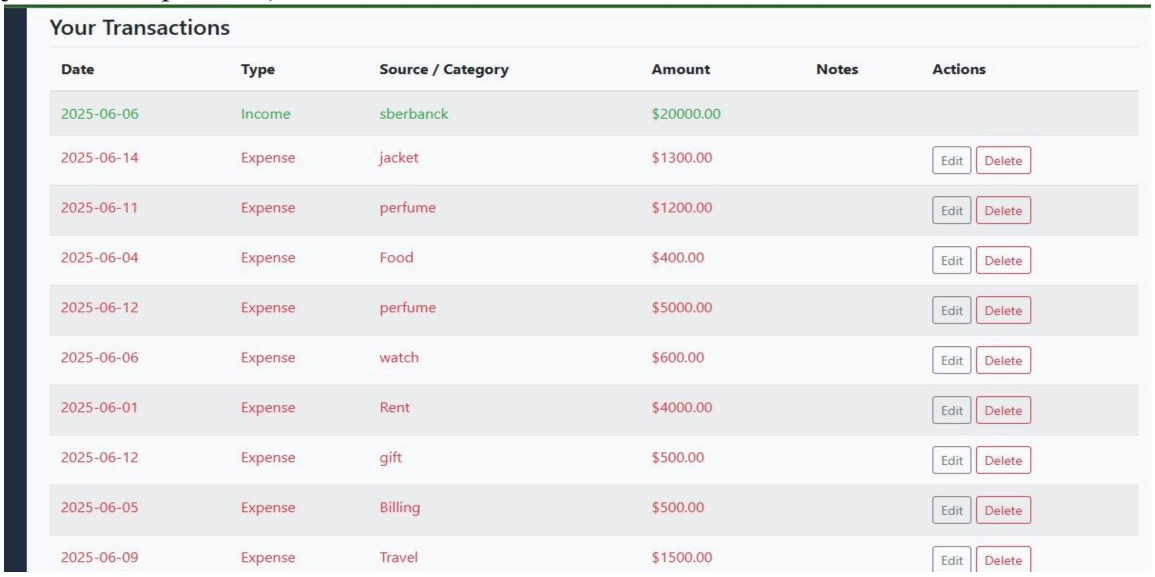
- Confirmation of successful Email Verification upon signup.


fintrack-project.onrender.com/accounts/confirm-email/Mg:1uQWw7:vNyXI5ws6amsuZAbKHmKJoyLRRMrCga3IlwqfBPR4jA/

Email Confirmation

Thank you SIDIMOHAMED, your email has been confirmed!

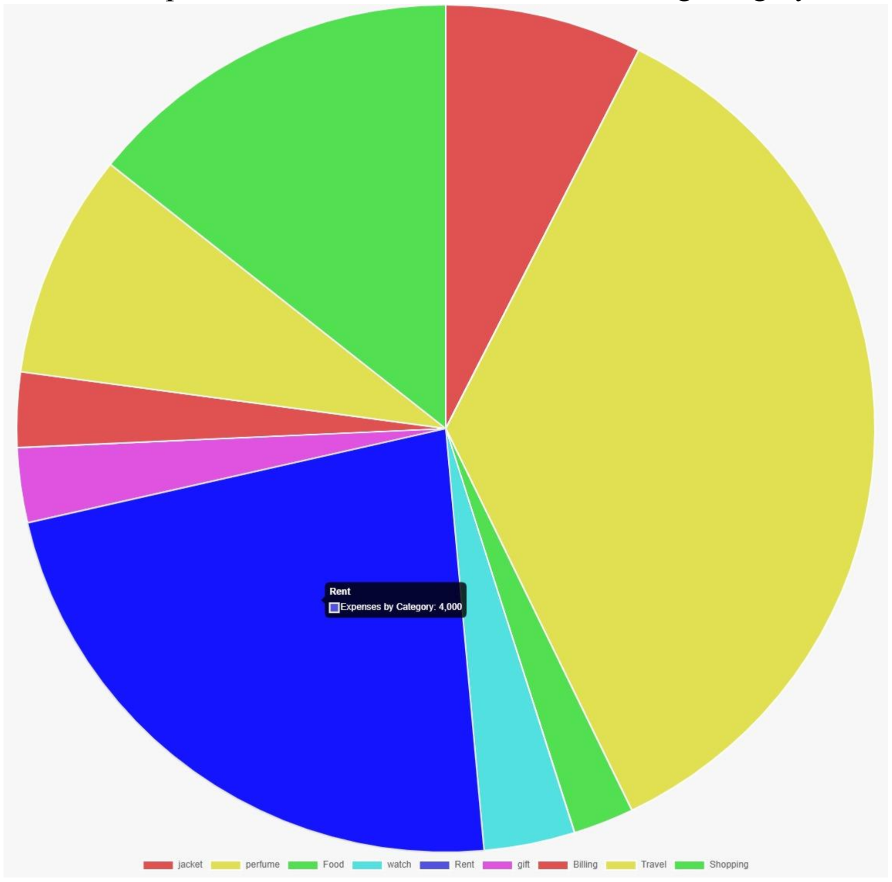
- A fully populated Transactions Table with real-world data entries (e.g., jacket, rent, perfume).



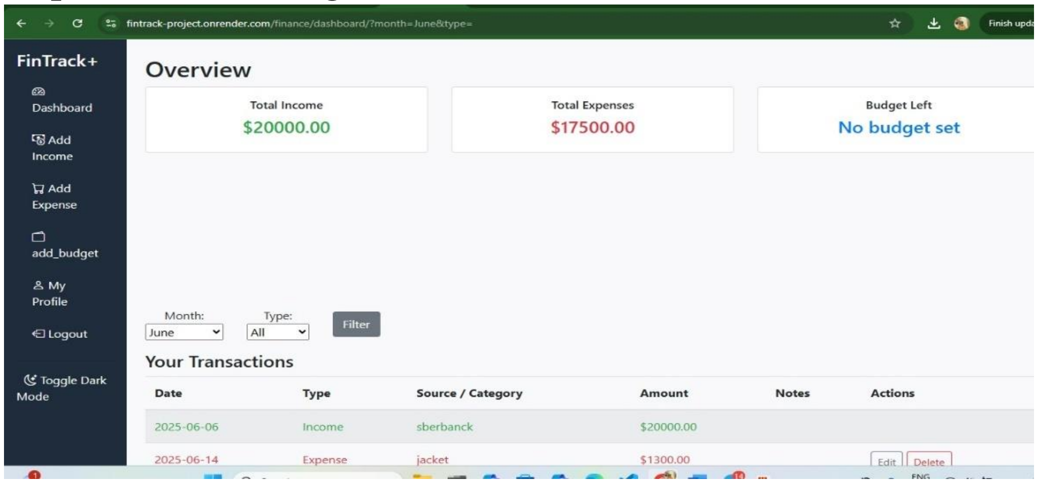
Your Transactions

Date	Type	Source / Category	Amount	Notes	Actions
2025-06-06	Income	sberbanck	$20000.00		
2025-06-14	Expense	jacket	$1300.00		Edit Delete
2025-06-11	Expense	perfume	$1200.00		Edit Delete
2025-06-04	Expense	Food	$400.00		Edit Delete
2025-06-12	Expense	perfume	$5000.00		Edit Delete
2025-06-06	Expense	watch	$600.00		Edit Delete
2025-06-01	Expense	Rent	$4000.00		Edit Delete
2025-06-12	Expense	gift	$500.00		Edit Delete
2025-06-05	Expense	Billing	$500.00		Edit Delete
2025-06-09	Expense	Travel	$1500.00		Edit Delete

- A responsive and accurate Pie Chart reflecting category-wise expense distribution.



- Overview cards summarizing Total Income, Total Expenses, and budget status.



Overview

Total Income	Total Expenses	Budget Left
$20000.00	$17500.00	No budget set

Month: June
Type: All

Your Transactions

Date	Type	Source / Category	Amount	Notes	Actions
2025-06-06	Income	sberbanck	$20000.00		
2025-06-14	Expense	jacket	$1300.00		Edit Delete

Conclusion

The development of the FinTrack+ application provided an invaluable opportunity to consolidate and expand my technical and analytical skills in full-stack web development. Throughout the project, I gained in-depth experience in designing, building, and deploying a secure and user-centric financial management system using Django, Bootstrap, and supporting tools.

One of the most significant learning outcomes was mastering the integration of backend logic with an intuitive and responsive frontend. By implementing features such as recurring transactions, budget tracking, and automated email alerts, I learned to address real-world needs through efficient database modeling, scheduled tasks, and user notifications. Moreover, handling user registration, authentication, and profile management deepened my understanding of Django’s authentication system and the practical use of Django Allauth.

The impact of this project lies in its utility: FinTrack+ offers users a clear and organized view of their financial activities, promotes responsible budgeting habits, and empowers informed financial decision-making through interactive visualizations and timely alerts.

From a personal and professional development perspective, this project allowed me to acquire and refine several key skills:

- Full-stack development with Django, including model-view-template (MVT) architecture.
- Database design and optimization, using SQLite for development and PostgreSQL for deployment.
- Integration of third-party libraries, including Chart.js and Django Crispy Forms, to enhance UI/UX.
- Deployment and configuration, using Docker and Render to launch a production-ready application.
- Security practices, including CSRF protection, secure email handling, and user session management.

In conclusion, this project not only showcased my ability to build a practical, scalable web application but also equipped me with the confidence and expertise required for professional software development and future academic endeavors.

References

1. Django Documentation. The Web framework for perfectionists with deadlines. Retrieved from: https://docs.djangoproject.com
2. Bootstrap Documentation. Build fast, responsive sites with Bootstrap. Retrieved from : : https://getbootstrap.com
3. Django-Allauth Documentation. Integrated set of Django applications addressing authentication, registration, account management. Retrieved from: https://django-allauth.readthedocs.io
4. Chart.js Documentation. Simple yet flexible JavaScript charting. Retrieved from: https://www.chartjs.org
5. Render Deployment Guides. Deploy your apps seamlessly with Render. Retrieved from: https://render.com/docs
6. Django REST Framework. Powerful and flexible toolkit for building Web APIs. Retrieved from: https://www.django-rest-framework.org
7. Whitenoise Documentation. Radically simplified static file serving for Python web apps. Retrieved from: https://whitenoise.evans.io
8. Python Official Documentation. The Python Language Reference. Retrieved from: https://docs.python.org
9. SQLite Documentation. Small. Fast. Reliable. Choose any three. Retrieved from: https://www.sqlite.org/docs.html
10. GitHub Guides. Hello World - Getting started with GitHub. Retrieved from: https://docs.github.com/en/get-started/quickstart/hello-world

Final Showcase

To conclude, this project demonstrates a practical and user-friendly solution for personal financial management. From budgeting and transaction tracking to interactive data visualization, FinTrack+ helps users stay organized and informed throughout their financial journey.

We invite you to explore the live application and dive into the source code for deeper insights into the implementation and architecture.

Live Demo: https://fintrack-project.onrender.com
GitHub Repo: https://github.com/Sidimohamedlemin/fintrack-project